|  |  |  |
| --- | --- | --- |
|  | **MANIPAL INSTITUTE OF TECHNOLOGY**  (A constituent Institute of MANIPAL UNIVERSITY)  **MANIPAL - 576 104, KARNATAKA, INDIA** |  |

Industrial Training

on

Semantic Search Engine using R

SUBMITTED

BY

Shashwat gupta 140905047 9620193750



Under the Guidance of:

Vivek Gupta  
Data Analytics Engineer

Data Science Practice

IMPETUS INFOTECH

**Abstract**

A semantic search engine is an improvised version of the traditional cosine based search engine where the cosine score is calculated using a query vector and another vector which constitutes of the tf-idf values for those words in the word-document table.

A semantic search engine is based on the concept of Latent semantic analysis. LSA is a technique in [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing), in particular [distributional semantics](https://en.wikipedia.org/wiki/Distributional_semantics), of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms.

Semantic search engines are more accurate since they not only return exact matches but also return semantically similar words such as cars and automobiles. My search engine was built on R studio as a shiny web app. It has the functionality of returning top 10 results from the total number of documents in the corpus.

Search engines have evolved since returning hard-match to the use of tf-idf values which was later modified because of exploitation by SEO experts. Cosine score matching and Semantic analysis is now used to return genuine and related documents.

R language was used throughout the project. Basics of R were learnt by using online tutorials and prescribed material by the supervisor. Smaller and easier projects were completed before starting this project. Important algorithms like Linear Regression, Clustering, Decision Tree and Random Forest were revised and implemented in R.

The documents were loaded from a CSV file into an R data frame. A posting list was created after removing stopwords and cleaning the Data. The count, tf , idf values were calculated using mathematical formulas. A term-Document occurrence matrix was made. Query is taken from the user and tf, idf, tf-idf values are calculated dynamically by the server code of the shiny app.

LSA is implemented using in-built function which uses the tf-idf matrix.

The returned matrix is then converted into text matrix. Query and LSA vector are created and then cosine function is used to match the query with the documents. Corresponding document numbers along with the Headline of the document are also returned to the UI element of the shiny App which displays the data as a data table output.

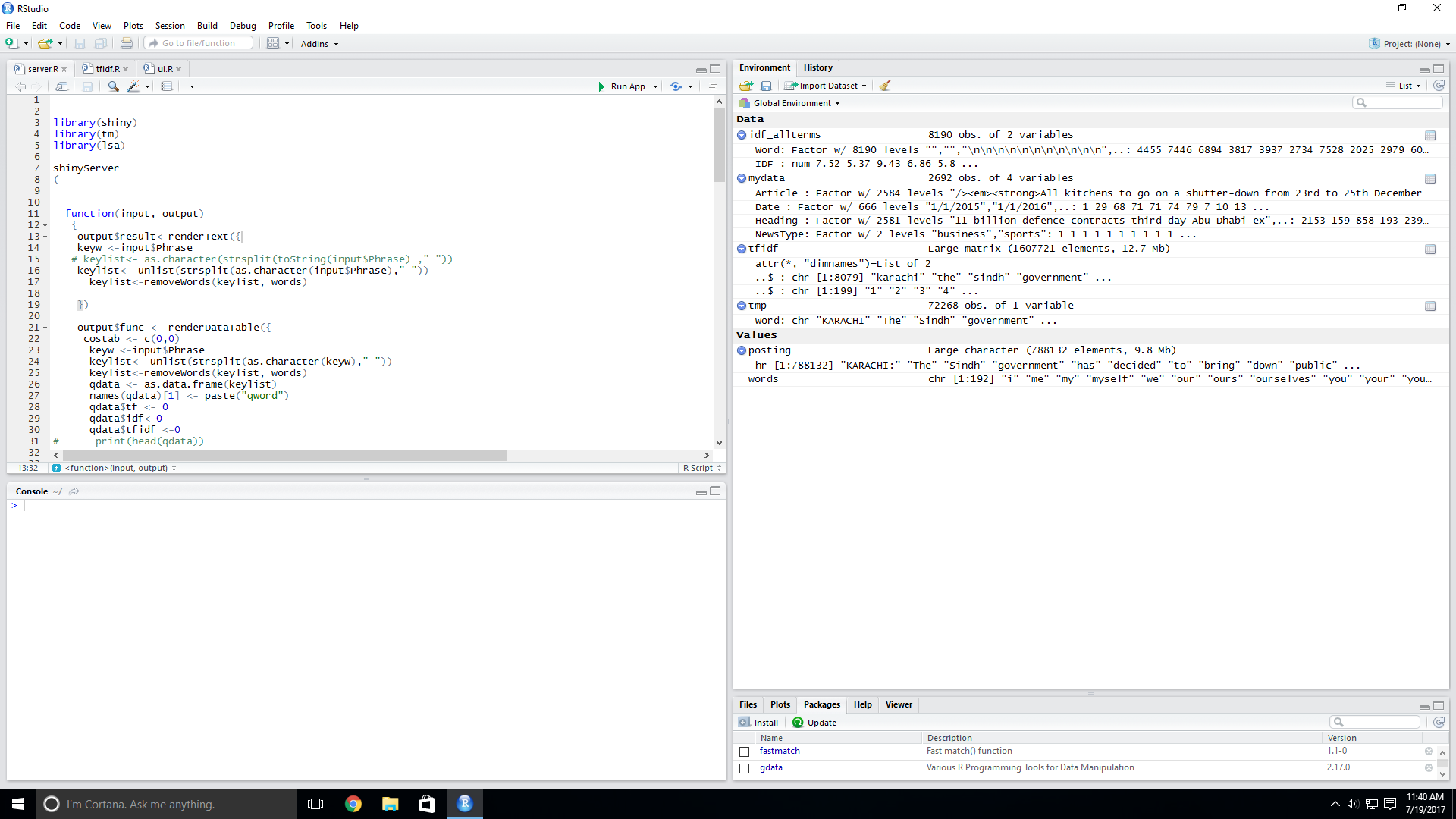
**Methodology**

Before starting to work on the search engine, R was learnt and practiced using online tutorial [1].

Additional knowledge of Information retrieval and Data Warehouse and Data Mining concepts was extremely advantageous.

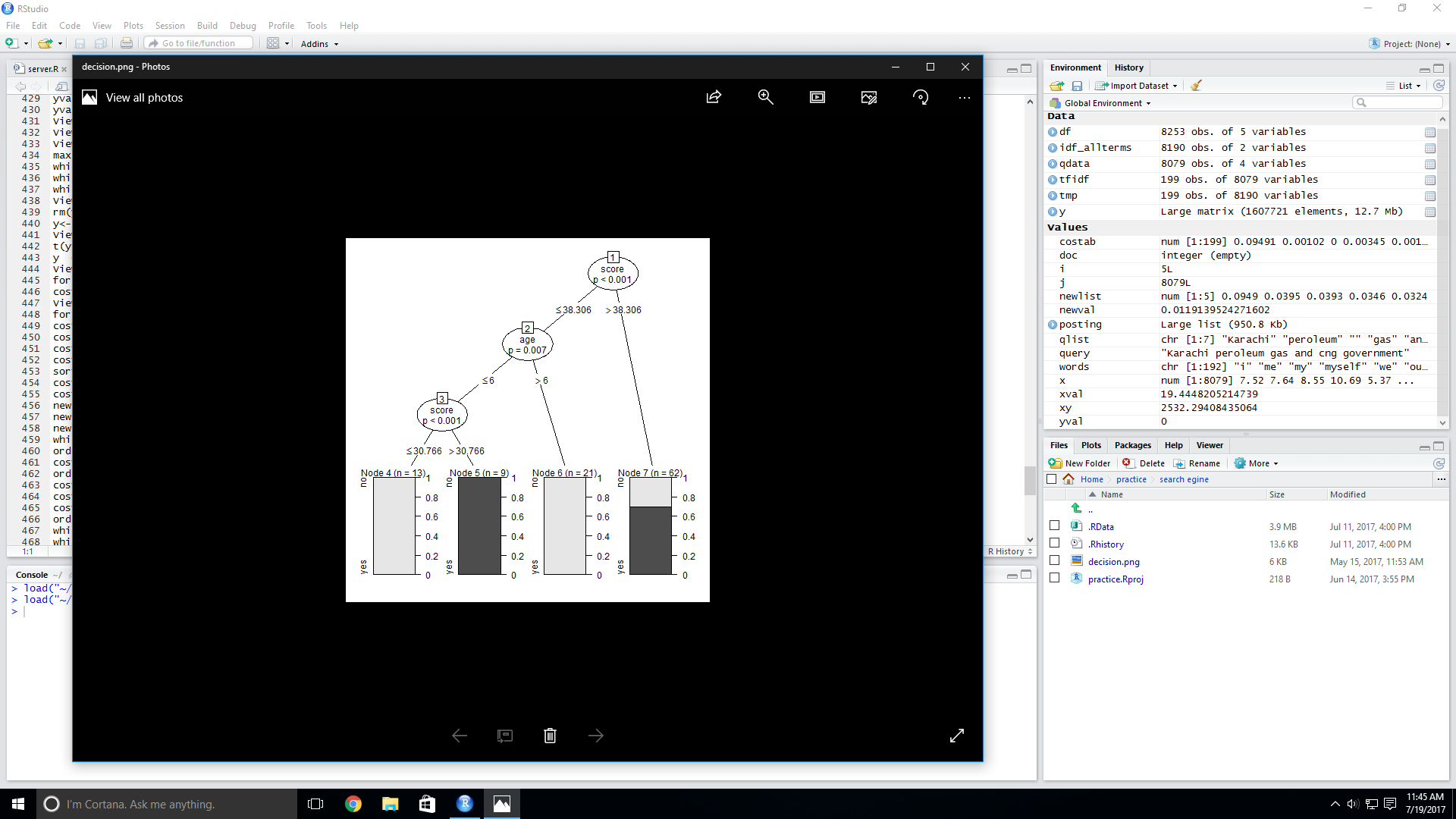
R studio is divided into 4 different sections for a simplified view.

* Code Section – Developer Area
* Console - Command Interface
* Global Environment – Contains all the Data Frames and Values
* Package Explorer – View all files and Packages



*Figure 1: R-studio Workbench*

Important algorithms like Linear Regression, Clustering, Decision Tree and Random Forest were revised [2] from and implemented in R.



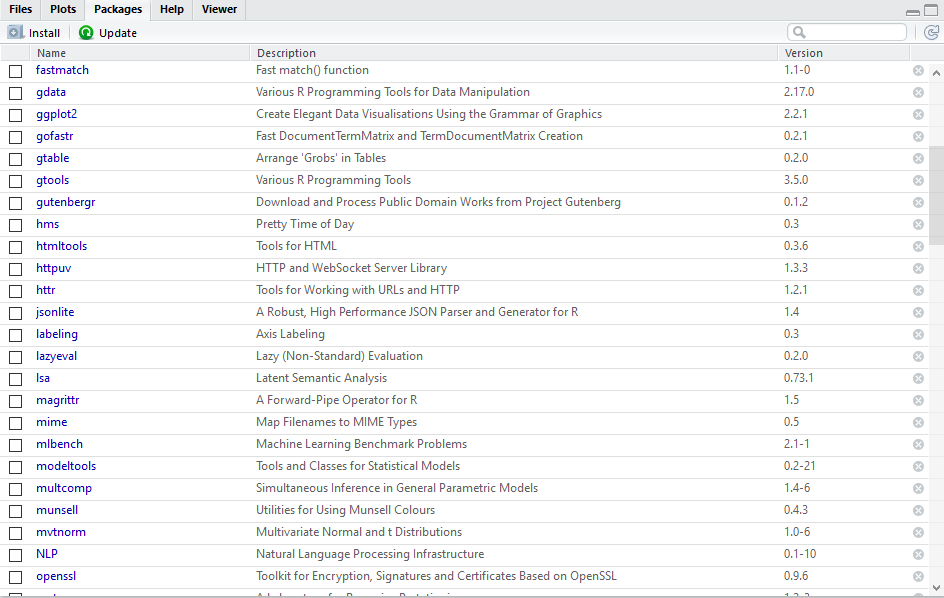
*Figure 2: Implementation of Decision Tree Algorithm*

Christopher manning [2], which was prescribed during our PE (Information Retrieval) was very useful since all the concepts needed to be revised before starting an actual project.

I found out that almost all Data Analytics projects which are carried out at impetus use algorithms like Linear Regression, Clustering or Random Forest.

I had some difficulty understanding Support Vector Machine (SVM) which is not covered in our course. Although packages can be installed for SVM and can be easily used but I faced a great deal of difficulty in understanding the underlying concept behind this Algorithm.

The lack of knowledge of this algorithm delayed my project by around a week and I had to rely upon the inbuilt function of the different packages like “lsa” , “ggplot2”,”tm-map” .

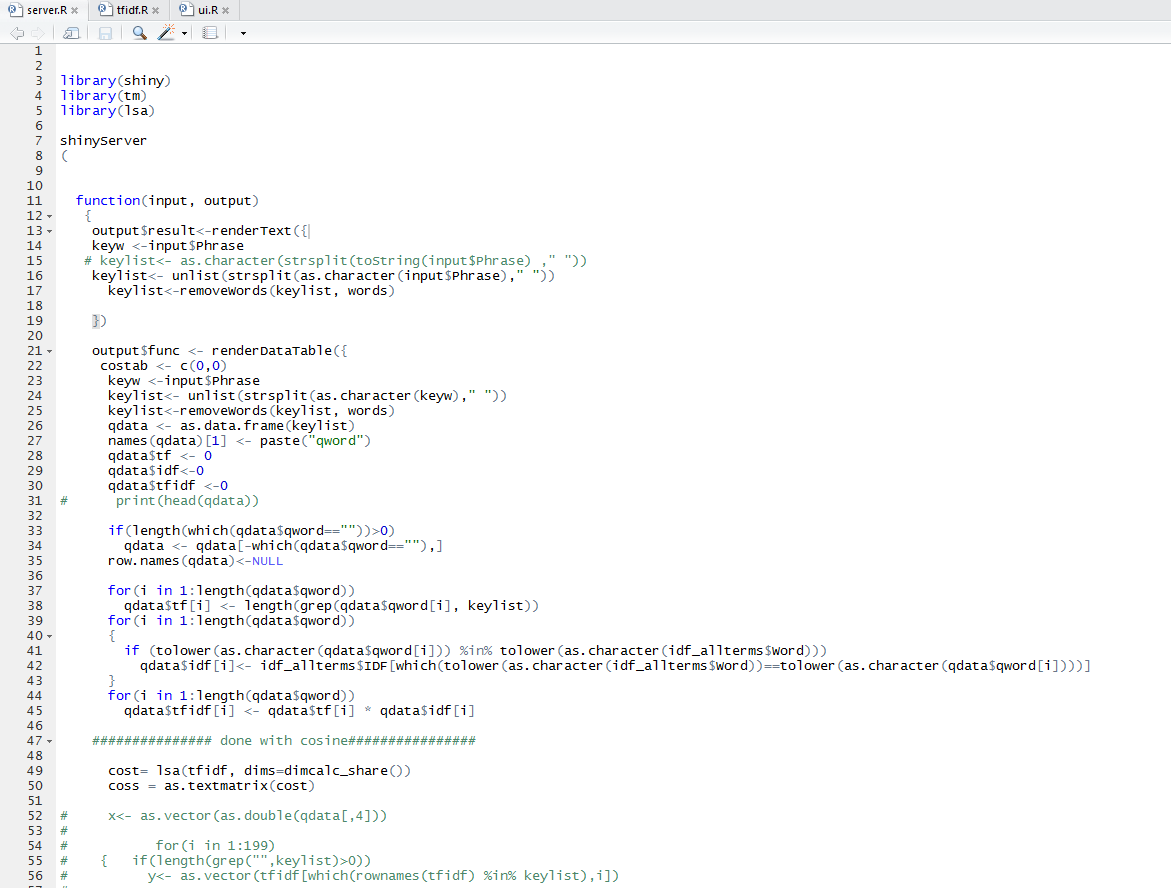


*Figure 3: Package Explorer Expanded*

After implementing small programs like the word counter in R, I was required to implement the same using the Shiny App interface which is a Graphical User Interface.

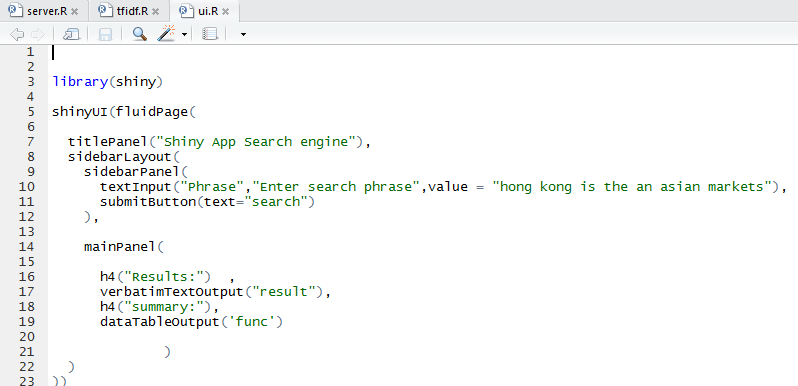
Shiny App has 2 different elements that need to be catered to:

* Server - All the backend code is written here. All dynamic calculations are taken care of by the server code.



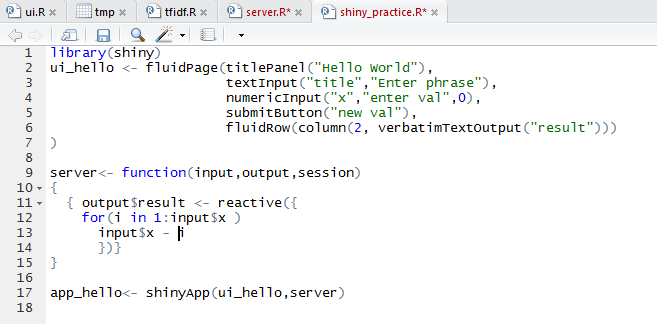
*Figure 4: Server Code Sample*

* User Interface – The UI code takes care the of the physical appearance of the webpage and how the different objects present on that page interact with the server code.



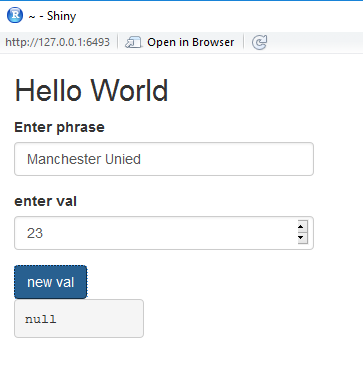
*Figure 5: UI code Sample*

A basic shiny App was implemented afterwards.



*Figure 6: Shiny App practice*

This code was written to practice the different elements/functionality that shiny app offers.

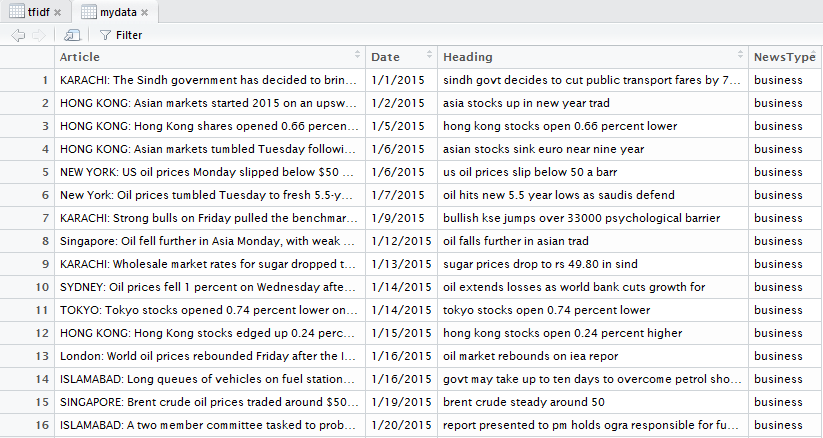


*Figure 7: Shiny App output*

This program also lets us decrease the “enter val” input value dynamically with every click of the “new val” button.

After getting the hang of the new language and implementing a few algorithms I was requested to work on my main project “Latent Semantic Search Engine using R”.

Firstly a corpus was downloaded from kaggle [3], which constituted of articles, date of publication, Headline and the Genre of the news and loaded into a new DataFrame called myData.

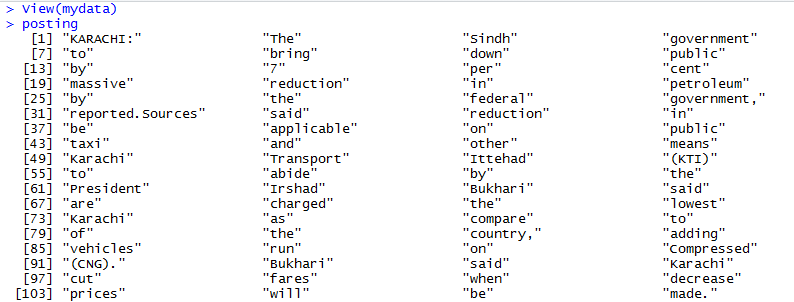


*Figure 8: Articles Dataset sample*

Irrelevant columns like Date and News Type were removed using setting the entire column to NULL. Secondly, a posting list needed to be created from all the documents after stop words and punctuations were stripped from the original documents in the DataFrame.

This was done using commands like –

1. Remove\_stopwords(as.data.frame(unique(removeWords(x$Article,words)))
2. Gsub(“[[:punct:]]”,” ”,df$word) -> df$word
3. Posting <- strsplit (as.String(as.character(x$Article)), “ “)



*Figure 9: Posting list data*

The new posting list was then copied to df as well as idf\_allterms data frames.

Idf\_allterms calculates the idf values for each word using:

**#saving the count in count variable**

for (i in 1:length(df$Word)) df$count[i]<-length(grep(df$Word[i],x$Article))

**#to find tf in each doc**

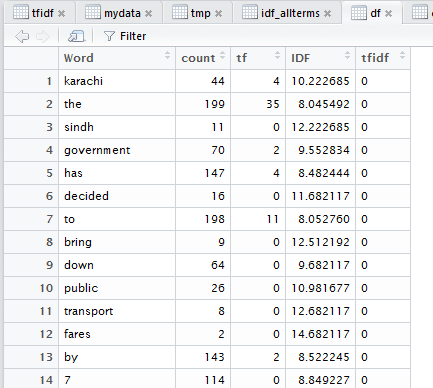
for(i in 1:length(df$Word))

df$tf[i] <- length(grep(df$Word[i],as.list(unlist(strsplit(as.character(x$Article[1])," ")[1]))))

**#to find the IDF value**

for(i in 1:length(df$Word))

df$IDF[i] <- log2(length(df$Word)/df$count[i])



*Figure 10: df data frame*

All calculations for all the terms of the cleansed posting list are done prior to query processing.

The tf-idf matrix is constructed between the terms and the document. This considers the total number of occurrences in a document rather than only the presence of the word in question at the moment.

**# to fill the table with frequency**

for(i in 1:length(unique(df$Word)))

for(j in 1:nrow(x))

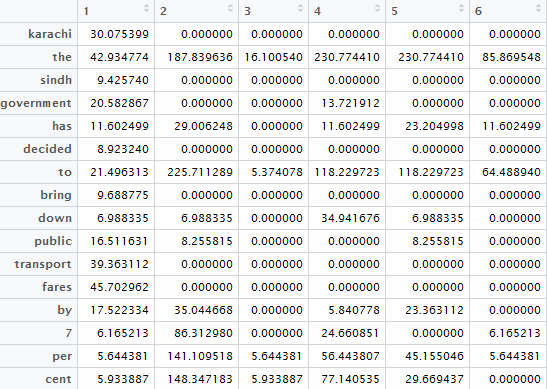
tmp[j,i] <- length(grep(unique(df$Word[i]),as.list(unlist(strsplit(as.character(x$Article[j])," ")))))

**#create a table for tf-idf**

tfidf<- as.data.frame(1:nrow(x))

for(i in 1:length(unique(df$Word))) tfidf[,i] <- 0

for(i in 1: ncol(tfidf)) tfidf[,i] <- tmp [ , i ]\* idf\_allterms$IDF[i]



*Figure 11: tf-idf all terms sparse matrix*

The tf, idf and tf-idf values of the query is calculated dynamically using:

for(i in 1:length(qdata$qword))

qdata$tf[i] <- length(grep(qdata$qword[i], keylist))

for(i in 1:length(qdata$qword))

{

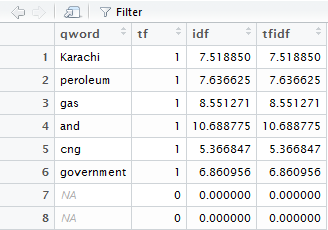
if (tolower(as.character(qdata$qword[i])) %in% tolower(as.character(idf\_allterms$Word)))

qdata$idf[i]<- idf\_allterms$IDF[which(tolower(as.character(idf\_allterms$Word))==tolower(as.character(qdata$qword[i])))]

}

for(i in 1:length(qdata$qword))

qdata$tfidf[i] <- qdata$tf[i] \* qdata$idf[i]



*Figure 12: query data tf-idf calculation*

To find the most relevant documents we first do latent semantic analysis on the tf-idf table and then convert it to text matrix format.

We then match the words with the keylist and store the result as a vector Y.

The query data is stored as X vector.

Both vectors are passed to the cosine function which return the cosine score between the two vectors. A new dataframe tmp is created and document numbers are retrieved from the articles table using:

x<- as.vector(as.double(qdata[,4]))

for(i in 1:199)

{ if(length(grep("",keylist)>0))

y<- as.vector(coss[which(rownames(tfidf) %in% keylist),i])

costab[i]<- cosine(x,y)

}

tmp <- as.data.frame(costab)

tmp$Docnum <- 0

tmp <- tmp[order(tmp$costab,decreasing = T),]

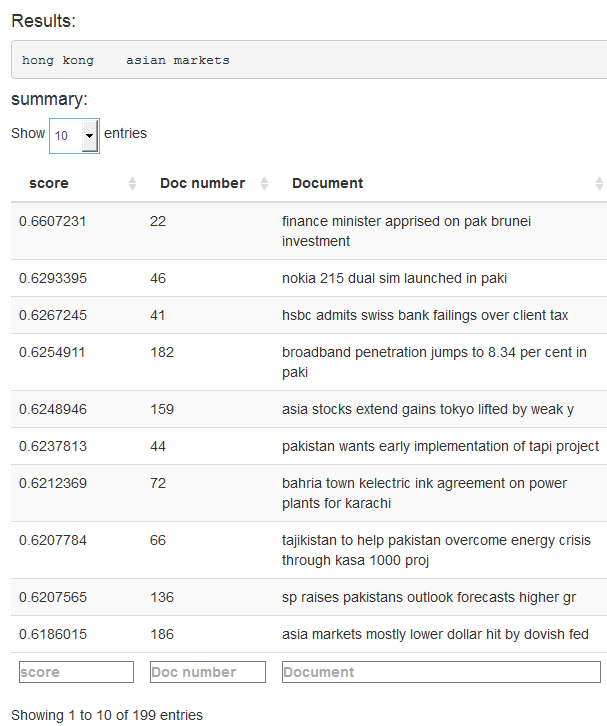
tmp$Docnum <- row.names(tmp)

row.names(tmp) <- NULL

tmp$Doc <- ""

tmp$Doc <- mydata$Heading[as.integer(tmp$Docnum)]

colnames(tmp)<- c("score","Doc number","Document")



*Figure 13: semantic search output*

**Conclusion**

The App successfully retrieved the top 10 results using latent semantic search.

Latent sematic search not only returned the exact match, but returned the most relevant document. We are also successful in returning the exact document number and the Headline for the same.

I tried to return the entire document but the WebApp did not allow me to return such big documents in the interface.

I also tried to implement the LSA using Support Vector Machine but due to insufficient clarity in the algorithm, I was unable to accomplish the task. I had to resort to the use of in-built functionality of the LSA package.

**References**

1. www.r-tutor.com
2. Introduction to Information Retrieval ,8th Edition by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze.
3. www.Kaggle.com
4. Impetus employee handbook

**Details of the Organization**

Impetus is a software solutions, products and services company headquartered in Los Gatos, USA with development centers in NOIDA, Indore, Gurgaon, and Bengaluru, India.

With more than 1600 employees globally, Impetus is focused on creating new ways of analyzing data for businesses—helping them gain key business insights across the enterprise.

They brought together a unique mix of Data Science capabilities and technology expertise across the Big Data ecosystem including Hadoop, NoSQL, NewSQL, MPP databases, machine learning, and innovative visualization.

They have recently been working on Data Warehouse Modernization. They have achieved success by using innovative techniques like Workload Migration and Data Blending. Workload Migration has offloaded traditional Data Warehouse to Big Data Warehouse and they were also able to blend diverse Data Sets and Accelerate the Time-to-Insight.

IMPETUS has 4 major Big Data Analytics Service Offerings:

* Consulting – Advisory services which help the client to define their Bi Data Strategy, blueprint and roadmap.
* Implementation – Agile implementation of advanced Big Data analytic applications.
* Data Science – Build advanced predictive analytical models and algorithms.
* Data Warehouse Modernization – Optimize and migrate data and applications from the enterprise Data Warehouse to the Big Data Warehouse.